ECE 8527 Final Project Summary

Kaijun Wang
Department of Statistical Science, Fox School of Business

Introduction: The data for this report comes from EEG signals used for seizure detection. Those signals are transformed using LFCC algorithm to obtain 26 features. The data is then divided into training set, test set and evaluation set. The training set contains 19320 observations, the development test set contains 2118 observations, and the evaluation set contains another 1174 observations, all of which are labeled by 0 or 1. The objective of this report is to construct a classifier for the data set with reasonable performance, this is evaluated on both training set and the development test set as well as the evaluation set. The training data and test data are both provided with a class label of 0 or 1 with evaluation set containing dummy class labels which are not used for building the prediction model. For this report, two classification methods are utilized to build the prediction model and we compared their performances on the dataset. The first method we use is the traditional K-Nearest Neighbor approach, for the choice of k we eventually set it to be 1000, as we discovered that this gives the most stable performance. For the second method, we implemented MLP neural networks and investigated how different topologies, optimizers, loss functions and batch sizes can affect the network performance. These comparison results are all shown in the attached figures.

These algorithms are implemented using a special build of Python 3.4 under Anaconda. We implemented the KNN method using the scikit-learn package provided at http://scikit-learn.org/stable/index.html. This package allows us to configure the KNN model in one single line while maintaining various customization options. For MLP neural network, we utilized the tools provided by Keras, which runs on TensorFlow backend. These tools also simplified our task of configuring and training the neural network models.

Algorithm No. 1 Description: For our first algorithm, we implemented the traditional K-nearest neighbor method. To investigate the effect of the value k on the model performance, we set it to range through 10 to 1000, for each value of k, we predicted the labels of the test set as well as training set, and recorded the error rates. These error rates are then used to compare the performance of each corresponding model. Those K-nearest neighbor models are trained using Euclidean distances with weights based on distance, this means the votes from other points are weighted by the inverse of their distance. For this dataset, we eventually picked k=1000 for our final model, as this number gives the most stable performance.

Algorithm No. 2 Description: For our second algorithm, we set up a MLP neural network consisting of 2 hidden layers, plus one input layer of 26 nodes and an output layer of 2 nodes. Each hidden layer is coupled with a 0.2 dropout rate to avoid overfitting. We also implemented autoencoders for pre-training the layers to improve accuracy. Each model is trained with 20 epochs. To understand how the many options for this model can the performance, several configurations are studied. each configuration is iterated 50 times to obtain an estimate for the distribution of corresponding error rates. The options we inspected include the effect of batch size, different loss functions, different types of optimizer, whether we normalize the data, and different setups for layer topologies. Due to the time consideration all the following comparisons are performed using a batch size of 120, but the final model will be trained on the optimal batch size. For activation functions, we used “Relu” for hidden layers and “sigmoid” for output layer.

Results: Our experiments were performed on a personal laptop with these specifications: (CPU: Intel Core i7-7700K 4.20GHz, Memory: 16 GB; GPU: NVIDIA GeForce GTX 1060, 6144 MB).

KNN classification result for both test set and the closed-loop are shown in Figure 1. Interestingly, in the cases we observed, the error rate seems to fluctuate around 0.45 for test set, and get more stable with increase number of neighbors. What’s more, the algorithm gives very accurate prediction for the training set, which is expected, since this method essentially assign most points in training set to whatever group it is from.

For deep neural network model, we first investigated the effect of batch size on the error rate, the models are trained with optimizer Adam, with loss function of mean square error. The layers are set up to be 26 inputs, 32 dense hidden layers, 52 dense hidden layers, then 2 outputs. Results are shown in Figure 2 We can see larger batch size causes over-fitting, which gives better performance in closed loop tests, but tend to worsen the error rates for open loop tests. We also tried some different loss functions by comparing the performance of mean square error and binary cross-entropy. Their performances are recorded on Figure 3, mean square error performs slightly better on the test data than binary cross-entropy. Next, we investigated some different layer topologies, with layer setups: a) [26,32,52,2], b) [26,26,32,2], c) [26,26,16,2], d) [26,16,10,2]. To clarify, setup a) would mean that first layer 26 nodes, second layer 32 nodes, third layer 52 nodes, etc. The error rates of these setups are recorded and shown Figure 4. Obviously, setup a) have the most prominent overfitting issue, but the rest of the topologies have a similar mean error rate. Layer setup b) was picked because it has the lowest minimum error rate among the simulations, while having reasonable variance and skewed to lower error rate. Lastly, the effect of data normalization is studied. We scaled and centered each column based on the means and variances of the training set. Again, each scenario is repeated 50 times, and the corresponding error rates are recorded, which is shown in Figure 6, Judging from the box plots, the normalization doesn't help much on the fit. Therefore, we choose not to perform the normalization process in the final model.

In the end, we built our neural network model as a MLP with four layers (two hidden layers): 26, 26, 32, 2, with 1632 parameters in total. The model uses MSE as loss function and for optimizer we use Adam, the model is trained with 20 epochs with batch size 20. The overall error rate for closed loop is between 0.33 to 0.36, and 0.36 to 0.40 for test data, which can be seen in Figure 7.

The results for both methods on the evaluation set are shown in Table 1. For KNN, the actual error rate on the evaluation set is 0.486, with Type I error as 0.637, and Type-II error as 0.335. For MLP neural network, the error rate is 0.460, with Type I error as 0.499, and Type-II error as 0.439. The overall error rates are about the same level, but KNN seem to have more false detections.

	Data Set		
Algorithm	**Train**	**Dev Test**	**Eval**
KNN	00.41%	45.56%	48.64%
MLP	38.40%	35.60%	46.00%

Table 1. Algorithm performance on training, testing and evaluation data, KNN with K=1000 gives best approximation for the training set, while MLP gives better results on testing and evaluation sets.

Conclusions: We studied the performance of two classification models on an EEG dataset, one of those is traditional k-nearest neighbor, another one is MLP neural network. In the end, their overall performances on the evaluation set are similar, both with considerably high error rate. The error rate for KNN is expected, but the error rate displayed for MLP is much higher than what the simulations on the test and training data indicated. The low performance again shows the overfitting problem we can have with neural networks. For KNN method, it might improve the results if we find some different distance measure and increasing k value, and for neural network, it is probably a good idea to implement higher dropout rate, as well as looking for better network structures.

Another interesting aspect to notice is that the actual error rate for the neural network results varies a lot (if no random seed is fixed aforehand). And from Figure 7 we can see that a random shift in error rate of about 1% ~2% is really a normal thing to happen. This gives rise to the question we've asked during earlier this semester that how much of an improvement on the error rate can be considered significant. Judging from the variance on the test set and the random guess error rate of 49.82%, we can say our model is significantly better than random guess.

Figure 1: Error Rate comparison: KNN

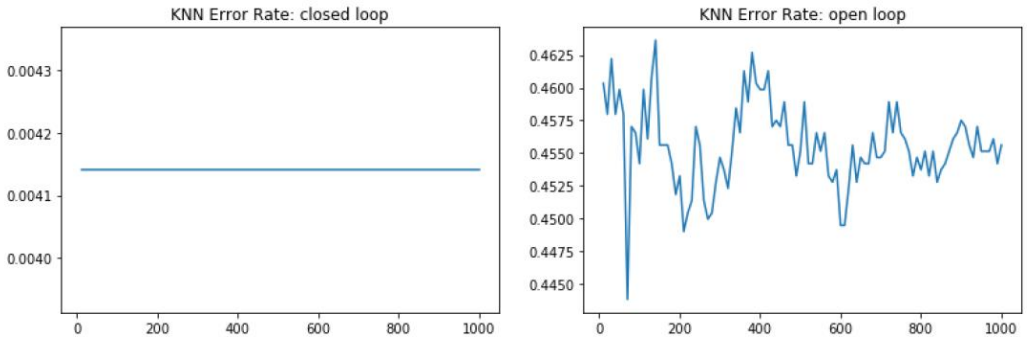

Figure 2: Error Rate comparison: MLP batch size

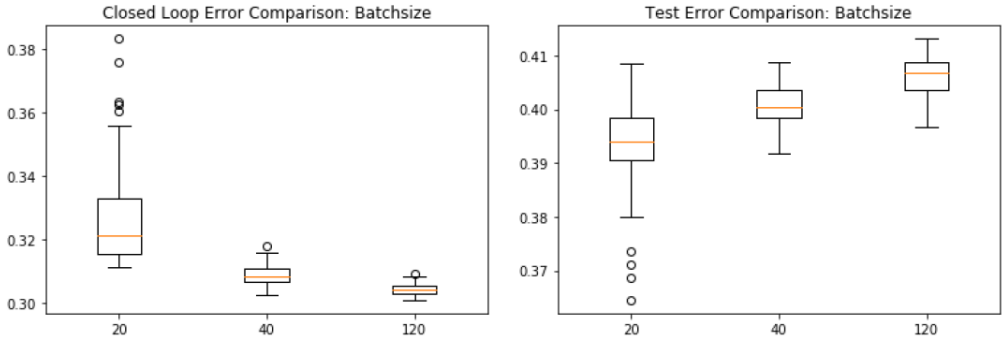

Figure 3: Error Rate comparison: MLP loss function

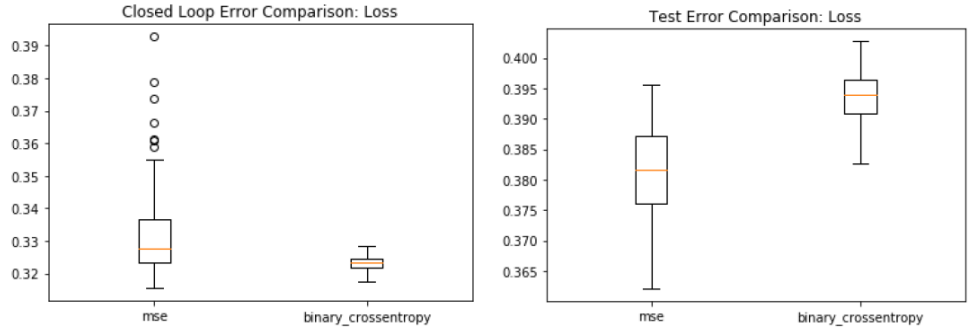

Figure 4: Error Rate comparison: MLP layers

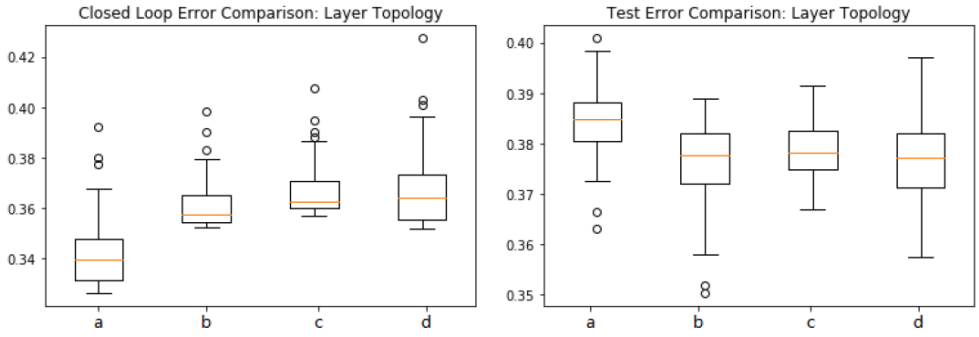

Figure 5: Error Rate comparison: Optimizers

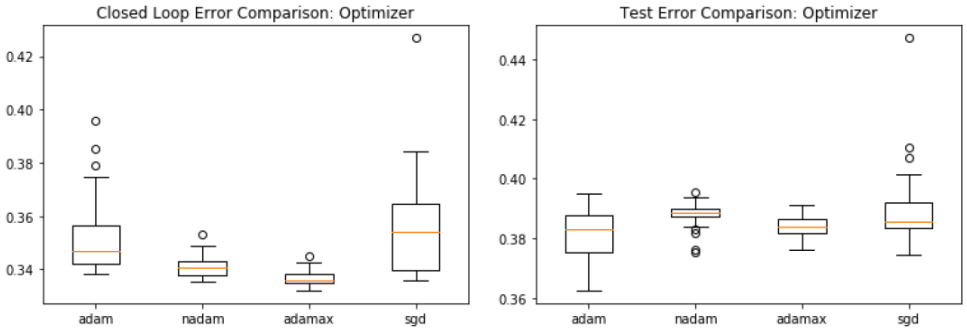

Figure 6: Error Rate comparison: Normalization

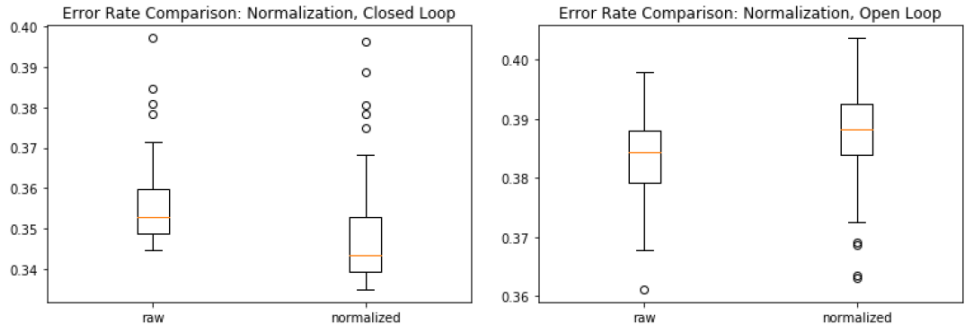

Figure 7: Error Rate: Overall